Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

CRT-2006-07

A TABU SEARCH HEURISTIC FOR THE GENERALIZED MINIMUM SPANNING TREE PROBLEM

by

Temel Öncan
Jean-François Cordeau
Gilbert Laporte

April 2006

Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7 Canada
Téléphone: (514) 343-7575 / Télécopieur: (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

HEC ÉCOLE POLYTECHNIQUE MONTRÉAL Université de Montréal

PUBLICATION
CRT-2006-07

A TABU SEARCH HEURISTIC FOR THE GENERALIZED MINIMUM SPANNING TREE PROBLEM

by

Temel Öncan[1]
Jean-François Cordeau[1]
Gilbert Laporte[1]

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

[1] Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7, and Canada Research Chair in Distribution Management, HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7 (oncan,cordeau, gilbert@crt.umontreal.ca)

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2006

106-04696/

Résumé

Cet article décrit une heuristique de recherche avec tabous basée sur la notion d'attributs pour le *problème d'arbre de recouvrement généralisé* (PARG) qui est NP-difficile. Étant donné un graphe dont l'ensemble des sommets est partitionné en sous-ensembles, le PARG consiste à concevoir un arbre de coût minimal reliant tous les sous-ensembles. Une heuristique de recherche avec tabous utilisant de nouveaux voisinages est proposée. Un ensemble étendu d'instances est généré et l'heuristique est comparée à des algorithmes génétiques récents. L'heuristique de recherche avec tabous obtient les meilleurs résultats pour toutes les instances. De plus, une adaptation est décrite pour une variante du PARG dans lequel chaque sous-ensemble doit être relié au moins une fois.

Mots-clés : arbre de recouvrement minimal généralisé, recherche avec tabous.

Abstract

This paper describes an attribute based tabu search heuristic for the *Generalized Minimum Spanning Tree Problem* (GMSTP) known to be NP-hard. Given a graph whose vertex set is partitioned into clusters, the GMSTP consists of designing a minimum cost tree spanning all clusters. An attribute based tabu search heuristic employing new neighborhoods is proposed. An extended set of TSPLIB test instances for the GMSTP is generated and the heuristic is compared with recently proposed genetic algorithms. The proposed heuristic yields the best results for all instances. Moreover, an adaptation of the tabu search algorithm is proposed for a variation of the GMSTP in which each cluster must be spanned at least once.

Keywords: generalized minimum spanning tree, tabu search.

1. Introduction

Consider an undirected graph $G = (V, E)$ where $V = \{v_1, \ldots, v_n\}$ is a vertex set partitioned into k clusters $V_1, \ldots, V_k$, $E = \{i, j : i \in V_p, j \in V_q, \text{ for } p, q = 1, \ldots, k, p \neq q\}$ is an edge set, and $C = (c_{ij})$ is a symmetric cost metric defined on E. The *Generalized Minimum Spanning Tree Problem* (GMSTP) consists of determining a minimum spanning tree (MST) containing exactly one vertex per cluster. Figure 1 illustrates a GMSTP solution with $k = 5$ clusters and $n = 13$ vertices.

Applications of the GMSTP arise, for example, in telecommunications, energy distribution and agricultural irrigation. The problem was first introduced by Myung et al. (1995) who have shown that it is NP-hard by reduction from the vertex cover problem. These authors also provided four integer programming formulations and developed a branch-and-bound algorithm. They have reported the exact solution of instances with up to 100 vertices. Feremans et al. (2002) have presented a theoretical comparison of eight GMSTP formulations, including those of Myung et al. (1995). In another article, Feremans et al. (2004) have

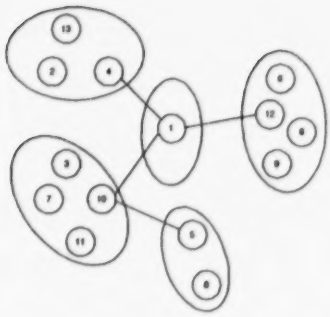

Figure 1: Example of a GMSTP solution

proposed several classes of valid inequalities and a branch-and-cut algorithm for this problem. They have employed a basic tabu search algorithm to obtain an initial feasible solution. Optimal solutions were reported for instances with up to 200 vertices. More recently, Pop et al. (2006) have proposed a new formulation of the GMSTP and a solution procedure that finds an optimal solution on instances with up to 240 vertices.

Feremans et al. (2004) have also proposed GMSTP adaptations of the well-known minimum spanning tree problem (MSTP) construction heuristics of Kruskal (1956), Prim (1957) and Sollin (see, e.g., Ahuja et al., 1993). They have tested the three adapted heuristics and found that the adaptation of the Kruskal algorithm is the best. Golden et al. (2005) have also experimentally studied the straightforward adaptations of the Kruskal, Prim and Sollin algorithms. They report that although for most instances the adaptation of the Kruskal algorithm gives the best solution values, the adaptation of the Sollin algorithm yields the best average percent deviation from the optimal solution value.

Golden et al. (2005) have also proposed a local search heuristic (LSH) and a genetic algorithm (GA) for the GMSTP which were tested on the TSPLIB instances generated by Feremans (2001). The authors report that on none of the instances does the LSH outperform the GA. With each of their LSH and GA, the authors have improved the best known solution values for TSPLIB instances with sizes $198 \leq n \leq 225$ and unknown optimal solution values. The best known solution values for these instances were reported by Feremans (2001).

A variation of the GMSTP involves the design of a tree spanning *at least* one vertex out of each cluster. This version of the problem is known as the L-GMSTP, where the letter L stands for the word *least*. The L-GMSTP was first discussed by Dror et al. (2000) who have proposed two formulations, a Lagrangian relaxation scheme, and five heuristics

including a GA. Haouari and Chaouachi (2006) have later improved the GA proposed by Dror et al. (2000) and have devised a stochastic heuristic using the PROGRES (PRObabilistic GREedy Search) method of Haouari and Chaouachi (2002). The authors have also analyzed three integer programming formulations and proposed an improved Lagrangian relaxation algorithm for the L-GMSTP.

In this paper, we present an attribute based tabu search (TS) heuristic for the GMSTP. For all benchmark instances with known optimal solution values, the TS algorithm reaches an optimal solution within reasonable CPU times. Moreover, among the benchmark instances with unknown optimal solution values it yields improved best known solutions on three instances. Encouraged by these preliminary results, we have generated new and larger GMSTP instances with sizes $229 \leq n \leq 783$ using the TSPLIB test set (Reinelt, 1991). Experimental results on these new instances are reported for the LSH and GA proposed by Golden et al. (2005) and for our TS algorithm. The same instances are also tested with a new GA recently developed by Golden et al. (2006). The TS algorithm yields the best solution values for all instances. Moreover, the TS algorithm is also adapted to handle the L-GMSTP. On the test instances generated by Dror et al. (2000), the TS algorithm for the L-GMSTP gives results comparable to those of the GA developed by Haouari and Chaouachi (2006).

The remainder of this paper is organized as follows. Section 2 introduces our TS algorithm for both versions of the GMSTP. Section 3 then presents a lower bounding procedure for the GMSTP. This is followed in Section 4 by computational results, and by conclusions in Section 5.

2. Tabu Search Heuristic

Tabu Search is an iterative local search method widely used in combinatorial optimization (Glover, 1986). Starting from an initial solution, the search method moves at each iteration from the current solution to the best one in a subset of its neighborhood, even if this causes a deterioration in the objective function value. To avoid cycling, solutions possessing some attributes of recently visited solutions are declared *forbidden* or *tabu* for a given number of iterations, called the tabu tenure. The algorithm stops whenever a preset criterion is satisfied. During the search, the algorithm maintains short term and long term memory structures.

When an attribute is declared tabu, all solutions possessing this attribute are implicitly declared tabu. However, some of these solutions may in fact never have been considered by the search. To remedy this, an aspiration criterion is defined to override the tabu status of a solution. One common aspiration criterion is to allow tabu solutions yielding better solution values than that of the best known solution. We refer to the book by Glover and Laguna (1997) for more implementation details on TS.

In our TS, we apply an extension of the aspiration level concept by associating an attribute to each vertex of the graph. Hence, the attribute set $A(s)$ of solution s is defined as the set of vertices selected for the solution s. We define the tabu rule as follows: if vertex i is removed from the minimum spanning tree, reinserting it into the tree is forbidden for the next θ iterations, where θ is the tabu tenure. The last iteration for which attribute i is declared tabu is denoted by β_i. The tabu status of an attribute can be revoked if it leads to a solution of smaller cost than that of the best solution identified having that attribute. The aspiration level γ_i of an attribute is initially set equal to $c(s_0)$ if vertex i belongs to the initial solution s_0, and to ∞ otherwise. At every iteration, the aspiration level of each attribute $i \in A(s)$ of the current solution is updated to $\min\{\gamma_i, c(s)\}$. Given a solution s, its neighbor solutions are denoted by $N(s)$. At each iteration, we consider a subset $T(s) \subseteq N(s)$ which consists of all solutions $\overline{s} \in N(s)$ reachable from s without incurring the risk of cycling, i.e., those that are not tabu or that satisfy the aspiration criterion.

2.1. Neighborhoods for the GMSTP

A solution s of the GMSTP is represented with an array of size k where each element stores one vertex i out of each cluster V_p. Hence, one can obtain a solution of the GMSTP by solving the MSTP for the set of vertices in the array. In order to solve the MSTP we apply the Kruskal algorithm which runs in $\mathcal{O}(|E| + k \log k)$ time.

For the GMSTP we propose the c-cluster neighborhood $N^c(s)$ defined as follows. First select c clusters at random. For each cluster V_p, replace the vertex $i_p \in V_p \cap A(s)$ with a different vertex $j_p \in V_p \backslash A(s)$. Then, reconstruct a solution with the set of selected vertices by using the Kruskal algorithm. The 1-cluster neighborhood $N^1(s)$ thus considers all possible $|V_p| - 1$ solutions as the neighborhood of s, while $N^2(s)$ considers $(|V_p| - 1) \times (|V_q| - 1)$ solutions, where V_p and V_q are the two selected clusters. Clearly, the larger the number of selected clusters, the wider the search space. In our TS algorithm we employ the $N^1(s)$, $N^2(s)$ and $N^3(s)$ neighborhoods.

Although $N^3(s)$ is more powerful than $N^1(s)$ and $N^2(s)$, it may require excessive CPU times to check all possible combinations. On the other hand, although the neighborhood $N^1(s)$ is the weakest, it may help to search the neighbor solutions faster than $N^2(s)$ and $N^3(s)$. Hence, considering this trade-off, we propose a hybrid strategy to employ the three neighborhoods in a balanced way. More precisely, we use them consecutively following the sequence $N^1(s)$, $N^2(s)$, $N^3(s)$, $N^1(s)$,... We first start with neighborhood $N^1(s)$ and apply it until there is no improvement in the solution for t_1 consecutive iterations. Then we switch to neighborhood $N^2(s)$ and use it until no improvement is obtained for t_2 iterations. Finally, neighborhood $N^3(s)$ is employed until there is no improvement for t_3 iterations and again we switch to neighborhood $N^1(s)$. This process is repeatedly performed until the algorithm satisfies the stopping criterion. We have chosen $t_1 = 250$, $t_2 = 175$ and $t_3 = 100$, which seemed to work best for most instances.

2.2. Neighborhoods for L-GMSTP

Unlike what is done for the GMSTP, a solution of the L-GMSTP is represented with a binary array of size n: the i^{th} element of this array is set equal to 1 if and only if the vertex i is in the solution. As explained by Dror et al. (2000), minimizing the cost of the spanning tree may require the inclusion of additional vertices other than those already in the solution.

Given a solution array of size n, let R denote the set of selected vertices in the solution, namely R consists of the vertices in $A(s)$. Let S denote the set of vertices that are not in the solution, namely S consists of the vertices in $V \setminus A(s)$. We can consider R as the terminal vertices and S as the Steiner vertices. To find a feasible topology for the L-GMSTP we need to solve a Steiner tree problem (see, e.g., Chopra and Rao 1994), with terminal vertices R and Steiner vertices S. For this purpose, we employ the insertion heuristic proposed by Takahashi and Matsuyama (1980) and which runs in $\mathcal{O}(|R|\, n^2)$ time.

The insertion heuristic starts by removing a vertex i from the set R and assigning it to a temporary set U. At each iteration, a vertex $j \in R$ closest to the set U is found. The set of edges in a shortest path from j to U is added to an edge list L. Then, the vertex j is removed from R and inserted into U. The set of Steiner vertices on the shortest path is also added to U. The heuristic algorithm stops when the set R is empty. The insertion heuristic outputs the set of selected vertices U and the set of selected edges L. The steps of the insertion heuristic are summarized below.

Insertion Heuristic (Takahashi and Matsuyama, 1980)
begin
Input R
Initialization Set $U = \emptyset$, $L = \emptyset$
Remove a vertex i from R and insert it into U
  while $R \neq \emptyset$ **do**
    Find a $j \in R$ closest to the set U
    Add to the edge list L the edges of a shortest path from j to U
    Remove vertex j from R and insert it into U
    Add to U the Steiner vertices on the shortest path from j to U
  endwhile
Output U **and** L
end

The definition of the c-cluster neighborhood $N^c(s)$ discussed in the previous section is slightly modified for the L-GMSTP. Again, we first select c clusters at random. For each selected cluster p, we then define the set Q_p of vertices from V_p that belong to the current solution, i.e., $Q_p = V_p \cap A(s)$. We then initialize the set R with the vertices in $A(s)$ that do not belong to any of the selected clusters. Finally, one vertex $i_p \in V_p \setminus Q_p$ is added to R for each selected cluster p, and the insertion heuristic is run from the set R. As a result, $N^1(s)$ considers $|V_p \setminus Q_p|$ solutions as the neighborhood of the current solution s, while $N^2(s)$ considers $|V_p \setminus Q_p| \times |V_q \setminus Q_q|$, where V_p and V_q are the two selected clusters. As for the GMSTP, we use three neighborhoods by following the sequence $N^1(s)$, $N^2(s)$, $N^3(s)$ and $N^1(s)$. Here, we set $t_1 = 25$, $t_2 = 20$ and $t_3 = 15$.

2.3. Attribute based tabu search algorithm

We now describe the steps of the TS algorithm which are the same for both the GMSTP and the L-GMSTP. However, the neighborhood definitions and the parameter settings are different.

2.3.1. Construction of an initial solution

In order to construct an initial feasible solution we randomly select a vertex from each cluster. For the GMSTP, we solve the MSTP for the set of selected vertices by using the

Kruskal algorithm. For the L-GMSTP, instead we run the Takahashi and Matsuyama (1980) insertion heuristic. The initial solution is set as the best solution s^* identified by one of these two algorithms.

2.3.2. Initialization of parameters

We first initialize the number of times attribute i has been added to the solution. This parameter is set equal to 1 for all vertices in the current solution (i.e., $\alpha_i = 1$ for all $i \in A(s)$, and $\alpha_i = 0$ for all $i \notin A(s)$). Then, the last iteration for which attribute i is declared tabu is set equal to 0 for all vertices (i.e., $\beta_i = 0$ for all i). The aspiration levels γ_i are set equal to the initial solution value $c(s)$ for all vertices in the current solution (i.e., $\gamma_i = c(s)$ for all $i \in A(s)$ and $\gamma_i = \infty$ for all $i \notin A(s)$). The number of times cluster j has been considered for the neighborhood search is set equal to 0 for all clusters (i.e., $\delta_j = 0$ for all $j = 1, \ldots, k$). Finally, the iteration counter λ is set equal to 0.

2.3.3. Neighborhood search

The first step of our neighborhood scheme is the cluster selection procedure which is crucial for the performance of the neighborhoods $N^1(s)$, $N^2(s)$ and $N^3(s)$, and hence to the performance of the TS algorithm. As a simple rule, we can randomly select clusters. However, during the run of the TS algorithm this may cause the algorithm to repeatedly visit the same clusters and hence the same search space without improving the solution. Especially, when the TS algorithm runs for a small number of iterations, some clusters are selected less often than others. Therefore, we propose a diversification scheme for the cluster selection considering the cluster visit frequencies, i.e., the number of times δ_j a cluster j has been selected. The cluster selection is performed with probabilities proportional to the inverse of the δ_j values. Hence, the lower the value of δ_j, the higher the probability of selecting cluster j. When all clusters have been visited an equal number of times, we apply the random selection rule for at most φ non-improving iterations. If the random selection of clusters has not yielded an improved solution for φ iterations, we again switch to cluster selection according to visit frequencies, and so on.

The next step after cluster selection is to evaluate a subset $T(s)$ of neighbor solutions. For this purpose, we first initialize $T(s) = \emptyset$. Then, for each neighbor solution $\bar{s}$ of the current solution s, if there exists an attribute $i \in A(\bar{s}) \setminus A(s)$ such that $\beta_i < \lambda$ or $c(\bar{s}) < \gamma_i$, we set $T(s) = T(s) \cup \{\bar{s}\}$. In other words, we add to $T(s)$ the neighbor solutions $\bar{s}$ such

that either $\overline{s}$ is non-tabu or the cost of $\overline{s}$ is lower than the aspiration level γ_i of one of the attributes $i \in A(\overline{s})$. To penalize the vertices frequently added to the solution, we define a penalized cost value for each solution $\overline{s}$ in the subset of neighborhoods $T(s)$ as follows:

$$p(\overline{s}) = \begin{cases} c(\overline{s}) + \phi c(s)\sqrt{kn} \sum\limits_{i \in A(\overline{s}) \setminus A(s)} \alpha_i/\lambda, & \text{if } c(\overline{s}) \geq c(s) \\ c(\overline{s}), & \text{otherwise.} \end{cases}$$

Here, ϕ is a factor used to adjust the intensity of the diversification, while the square root factor $\sqrt{kn}$, which was first proposed by Taillard (1993) in the context of the vehicle routing problem, is used to compensate for instance size. Finally, considering all solutions $\overline{s}$ in $T(s)$ we identify the solution s' with minimum penalized cost value $p(s')$.

2.3.4. Parameter update

Considering the new solution s' we update the following parameters. First, we update the tabu status of the removed vertices $i \in A(s) \setminus A(s')$ in order to forbid them from reentering the solution for at least θ iterations (i.e., $\beta_i = \lambda + \theta$). Second, for each attribute $i \in A(s') \setminus A(s)$, the number of times attribute i has been added to the solution is increased by one (i.e., $\alpha_i = \alpha_i + 1$). Third, if the new solution is better than the current best solution (i.e., $c(s') < c(s^*)$) we update the current best solution (i.e., $s^* = s'$). Finally, for each vertex of the new solution we update its aspiration level (i.e., $\gamma_i = \min\{\gamma_i, c(s')\}$ for all $i \in A(s')$).

2.3.5. Stopping criterion

Several stopping criteria can be adopted for the TS algorithm such as a time limit, an iteration limit, or an absence of improvement for a given number of iterations. We choose our criterion so as to achieve a balance between computation time and solution quality. The TS algorithm stops when the iteration counter λ reaches the iteration limit η.

To the best of our knowledge, only Feremans et al. (2004) have proposed a TS for the GMSTP. This heuristic is employed within their branch-and-cut algorithm to determine an initial feasible solution. Their TS algorithm works as follows. Randomly select a vertex $i_p \in V_p$ in the current solution and replace it with another non-tabu vertex $j_p \in V_p$ yielding the best solution. Then the selected vertex j_p is declared tabu for $\theta = 8$ iterations and the algorithm is run for $1000n$ iterations. Our TS heuristic differs from the one proposed by Feremans et al. (2004) in several respects: we assign an attribute to each vertex, we use a larger, hybrid neighborhood, and we use an additional diversification procedure in the cluster selection process.

3. Lower Bounding Procedure

To compute lower bound values for the GMSTP we use the directed multi-commodity flow formulation proposed by Myung et al. (1995). In this formulation, for each cluster $l = 2, \ldots, k$, a commodity l is sent from cluster 1 to cluster l which receives exactly one unit of this commodity.

We define the following variables: continuous variables f_{ij}^l equal to 1 if and only if commodity l flows from vertex i to vertex j; binary variables y_i equal to 1 if and only if vertex i is in the solution; and binary variables w_{ij} equal to 1 if and only if vertices i and j are connected. Note that intra-cluster arcs and variables are not defined. The formulation is the following:

$$\text{Minimize} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} w_{ij} \tag{1}$$

subject to

$$\sum_{i \in V_l} y_i = 1 \qquad l = 1, \ldots, k \tag{2}$$

$$\sum_{j=1}^{n} f_{ij}^l - \sum_{j=1}^{n} f_{ji}^l = y_i \qquad i \in V_1, l = 2, \ldots, k \tag{3}$$

$$\sum_{j=1}^{n} f_{ij}^l - \sum_{j=1}^{n} f_{ji}^l = -y_i \qquad i \in V_l, l = 2, \ldots, k \tag{4}$$

$$\sum_{j=1}^{n} f_{ij}^l - \sum_{j=1}^{n} f_{ji}^l = 0 \qquad i \notin V_1 \cup V_l, l = 2, \ldots, k \tag{5}$$

$$\sum_{i \notin V_l} \sum_{j \in V_l} w_{ij} \leq 1 \qquad l = 2, \ldots, k \tag{6}$$

$$0 \leq f_{ij}^l \leq w_{ij} \qquad i, j = 1, \ldots, n; l = 2, \ldots, k \tag{7}$$

$$y_i, w_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n. \tag{8}$$

Constraints (2) ensure that exactly one vertex is chosen from each cluster. Constraints (3) state that one unit of each commodity leaves cluster 1 while constraints (4) state that one unit of commodity l enters each cluster $l = 2, \ldots, k$. Constraints (5) enforce flow conservation at intermediate nodes. Finally, constraints (6) ensure that there exists at most one incoming connection into each cluster. Notice that constraints (3)-(5) prevent cycles in the solution.

This formulation contains a polynomial number of variables and constraints. For small instances, its linear programming relaxation can thus be solved directly by any commercial solver. In our experiments we have used CPLEX 9.0.1 to solve these relaxations.

4. Computational Experiments

We now present the details of our TS algorithm which was implemented in C++ and run on a Pentium IV PC with a 3GHz CPU.

4.1. Parameter calibration

The fine tuning of parameters used within the TS heuristic must be carefully handled in order to obtain good results. For the GMSTP, we calibrated the parameters considering the TSPLIB instances with $198 \leq n \leq 226$.

We have first fixed $\theta = 7$, $\varphi = 100$ and $\eta = 30000$ and performed tests on all instances using different values of parameter ϕ (which is used to adjust the intensity of the diversification) in the interval $[0.01, 0.1]$ with increments of 0.01. We observed that for most of the instances the best value of the parameter ϕ lies in the interval $[0.03, 0.08]$. Hence we have set $\phi = 0.05$ as the most appropriate value.

To determine the most appropriate value for θ we have performed experiments by fixing other parameters (namely, $\phi = 0.05$, $\varphi = 100$ and $\eta = 30000$). Generally, setting parameter $\theta = 7$ seems to be appropriate for most of the TSPLIB instances with known optimal solution values. However, the proper value for this parameter also seems to depend on instance size. We thus performed experiments by choosing the value of θ from the interval $[\lceil 3 \log_{10} n \rceil, \lceil 5 \log_{10} n \rceil]$, where $\lceil x \rceil$ stands for the nearest integer greater than or equal to x. For most of the instances we observed that when the tabu duration θ is fixed to $\lceil 4 \log_{10} n \rceil$ we obtain better results. However, according to our preliminary tests we observed that varying the tabu tenure throughout the search yields better solutions than choosing a fixed value. Hence every time a vertex is declared tabu, we randomly select the tabu tenure from the interval $[\lceil 4 \log_{10} n \rceil - 2, \lceil 4 \log_{10} n \rceil + 2]$.

In order to control the diversification of the cluster selection process, we use the parameter φ. In this phase we successively apply two rules: random cluster selection and cluster selection according to frequency. At the start of the algorithm the clusters are selected randomly. Then, after φ iterations without improving the solution value, we switch to the

cluster selection by visit frequency. In our preliminary experiments, we observed that the most appropriate value for φ is 50. For values larger than 50 we have observed that the algorithm wastes too much time by selecting the recently visited clusters. On the other hand when we set lower values, we have observed that the clusters are visited sequentially since the visit frequencies of clusters become almost equal to each other.

To reach a balance between solution quality and CPU time we have set $\eta = 75n$ iterations with an upper bound of 40000 in order to limit the CPU time on larger instances.

The parameters of the TS algorithm for the L-GMSTP are fine-tuned as for the GMSTP. To calibrate our parameters we performed preliminary experiments on the instances provided by Dror et al. (2000). For the L-GMSTP, we set $\phi = 0.05$, $\varphi = 25$, $\eta = 20n$ as the most appropriate values. The tabu tenure parameter θ is randomly chosen from $[\lceil 4 \log_{10} n \rceil - 3, \lceil 4 \log_{10} n \rceil + 3]$.

4.2. The performances of the LSH and the TS

The LSH devised by Golden et al. (2005) is a simple and efficient heuristic. While it depends on the random initial solutions and random cluster visit order, it performs quite well for most of the small size instances. However, for larger instances we observed important variations in solution quality. One reason of this variation may be explained by its greedy nature and the random decisions made during the course of the algorithm. The steps of the LSH algorithm, which are repeated 500 times, are as follows.

1. Randomly select a single vertex from each cluster. If the subgraph defined by the edges between the selected vertex is connected, apply any one of the MST algorithms to this subgraph. Otherwise, repeat Step 1.

2. Randomly define an order in which clusters will be searched.

3. By following the order defined in Step 2, repeat the following steps until K sequential cluster visits result in no improvement:

   - while visiting a cluster, consider each node in the cluster as a replacement for the current node in this cluster; compute the cost of the solution for each possible replacement.
   - among the solutions computed in the previous step, identify the one giving the greatest improvement; if there is an improvement, implement it.

To better illustrate the performance of the LSH and the strengths of the neighborhoods used in the TS heuristic, we have tested them on a GMSTP benchmark instance. Figures 2 to 5 depict the best solution values obtained during the runs of the LSH and TS algorithms using only the $N^1(s)$, $N^2(s)$ and $N^3(s)$ neighborhoods for the *Kroa200* instance generated with the center clustering procedure. As can be observed, the LSH reaches the final value with an unstable curve and the shape of the curve reveals the random nature of the LSH. However, the curves obtained by the TS algorithm separately using the neighborhoods $N^1(s)$, $N^2(s)$ and $N^3(s)$ smoothly converge to the optimal value. As can be observed from Figures 3 to 5, the larger the neighborhood, the smoother the shape of the curve.

The TS algorithms with $N^1(s)$, $N^2(s)$ and $N^3(s)$ reach an optimal solution at iterations 3155, 2938 and 542, respectively. Moreover, they reach a solution value below 12000, at iterations 1190, 426 and 260, respectively. These results show the strengths of the neighborhoods. However, the TS algorithms with $N^1(s)$, $N^2(s)$ and $N^3(s)$, reach an optimal solution within 5, 7 and 14 seconds, respectively. This was to be expected since the larger the neighborhood, the more costly its exploration will be.

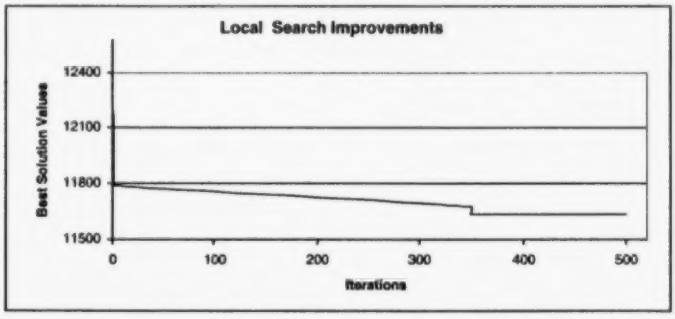

Figure 2: Best bounds vs. iterations for LSH on the Kroa200 instance

4.3. Extended TSPLIB instances for the GMSTP

We have generated an extended set of instances from the TSPLIB with sizes $229 \leq n \leq 783$ by following the method used by Feremans (2001). The new set of test instances can be downloaded from the site `http://www.hec.ca/chairedistributique/data`.

For each instance we have applied the center clustering procedure devised by Fischetti et al. (1997). In this cluster generation process, the number of clusters is fixed to $k =$

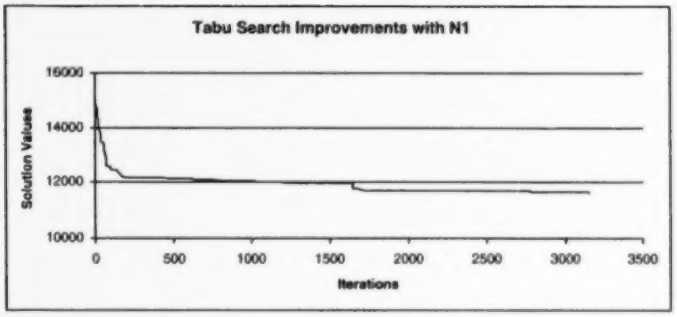

Figure 3: Best bounds vs. iterations for TS with $N^1(s)$ on the Kroa200 instance

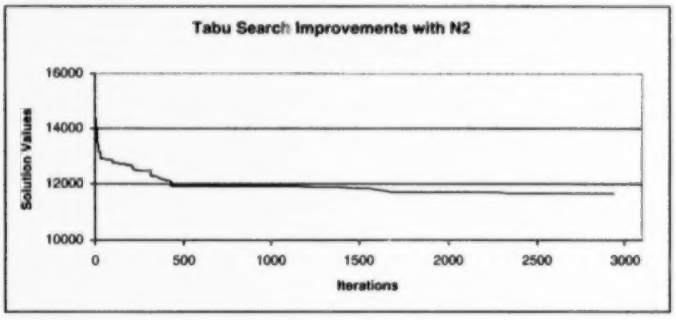

Figure 4: Best bounds vs. iterations for TS with $N^2(s)$ on the Kroa200 instance

$\lceil |n|/5 \rceil$. First, k center vertices are determined by considering k vertices that are as far as possible from each other. Then, the remaining $n-k$ vertices are assigned to the closest center vertices and k clusters are obtained. As explained by Fischetti et al. (1997) the grid clustering procedure is developed for instances with known geographical coordinates (x_i, y_i) for $i = 1, \ldots, n$. Considering all these coordinates, let x_{min}, y_{min} and x_{max}, y_{max} denote the minimum and maximum of the x and y coordinates, respectively. Then, the rectangle with coordinates (x_{min}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{max}) and (x_{min}, y_{max}) is subdivided into $ng \times ng$ grids such that each cell has width $(x_{max} - x_{min})/ng$ and height $(y_{max} - y_{min})/ng$. In this procedure, ng is defined as the smallest integer such that CLUSTER$(ng) \geq n/(\mu)$, where CLUSTER(H) is the number of nonempty clusters corresponding to the $H \times H$ grids. The grid clustering procedure is applied to all of the new instances except *pa561* and *si535* since for these two instances we only know the inter-vertex distances. However, for all other instances we have applied both center clustering and grid clustering for $\mu = 3$, $\mu = 5$, $\mu = 7$

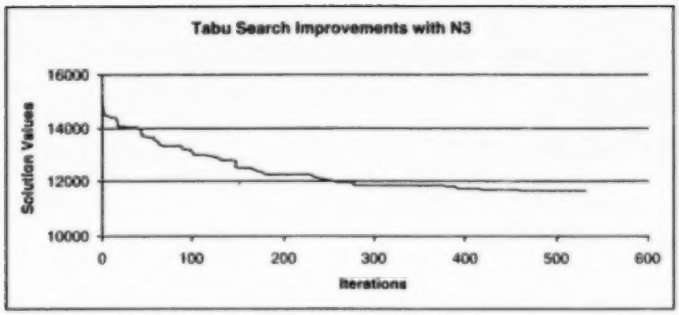

Figure 5: Best bounds vs. iterations for TS with $N^3(s)$ on the Kroa200 instance

and $\mu = 10$.

We would like to note a correction on the TSPLIB instances generated by Feremans (2001) for the GMSTP. For all these instances, Feremans (2001) has used the Euclidean distance (L_2 metric). However, for the *att* type instances and for the *geo* type instances, pseudo-Euclidean functions and geographical distance functions must be respectively used to compute the distances (Reinelt, 1991). Hence, the cost values for the TSPLIB instances *spain47*, *europ47*, *gr96*, *gr137*, *gr202* and *att48* differ from the correct values. However, we believe this small difference does not significantly affect the results reported in earlier articles.

4.4. Summary of the computational experiments

We provide in Table 1 the computational results for the LSH and GA proposed by Golden et al. (2005), as well as our results using TS, on the TSPLIB instances with sizes $198 \leq n \leq 226$. The lower bound values are computed using the linear programming relaxation of the formulation of Myung et al. (1995). These are tighter than the previous best lower bounds reported by Golden et al. (2005). Unfortunately, we were not able to compute the lower bound values for larger instances with the linear programming relaxation of the formulation. The results in columns **LSH** and **GA** are taken from Golden et al. (2005). Note that the TS heuristic improves the best known upper bound for three instances: *gr202* and *ts225* obtained with center clustering, and *ts225* obtained with grid clustering $\mu = 5$. Moreover, for all other instances the TS heuristic yielded bounds at least as good as those of LSH and GA. These preliminary results encouraged us to compare our TS algorithm on larger instances. Hence, we generated new larger TSPLIB instances for the GMSTP.

In Tables 2 to 6 we report the experimental results on the new TSPLIB instances for the GMSTP. The experiments with LSH and GA were separately performed by Golden et al. (2006) on a workstation with a 2.66 GHz Xeon processor and 2 GB RAM. In the **LS** column we report results obtained with the LSH of Golden et al. (2005). In the **GA1** column we provide the results obtained for the first version of the GA developed by Golden et al. (2005). The **GA2** column is for the improved version recently developed by Golden et al. (2006). In the last column, we present our results obtained with the TS heuristic. As can be observed, for all these new instances the TS heuristic yields the best solution values which are indicated in bold-face. We cannot directly compare the CPU times of our TS algorithm with those of the GAs by Golden et al. (2006), but one can argue that they are similar.

In Table 7 we present our experiments with the adaptation of our TS algorithm to the L-GMSTP case on instances generated by Dror et al. (2000). The optimal solutions are reported in Feremans (2001). In the column **GA**, we report the experimental results reported by Haouari and Chaouachi (2006) using their GA. These experiments were performed on a Pentium IV 2.4 GHz PC. In the column **TS**, we report the results obtained with the TS algorithm for the L-GMSTP. As can be observed, in only two out of 20 instances (i.e., DHC19 and DHC20) does the GA yield better solutions. The TS heuristic is clearly more costly than the GA, but the attribute based TS heuristic remains promising for the L-GMSTP.

5. Conclusion

We have developed an attribute based TS heuristic for the GMSTP and the L-GMSTP, using new neighborhoods. A new and larger set of test instances for the GMSTP was generated. The TS heuristic, two recently proposed state-of-the-art GAs and an LSH were tested on the new GMSTP instances. We have observed that the TS yields the best solutions. We have also adapted our TS algorithm to the L-GMSTP. Compared to a recent GA, the TS yields high quality results for the L-GMSTP.

Acknowledgments

This research was partially funded by the Canadian Natural Sciences and Engineering Research Council under grants 227837-04 and OGP0039682. The first author acknowledges the post-doctoral grant of "TÜBİTAK-BAYG Yurt Dışı Doktora Sonrası Araştırma Burs

Programı (NATO-B1)". The authors thank Bruce L. Golden, S. Raghavan and Daliborka Stanojević from the University of Maryland for performing additional experiments on our new data set.

References

Ahuja, R.K., T.L. Magnanti, J.B. Orlin. 1993. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, Englewood Cliffs, NJ.

Chopra, S., M.R. Rao. 1994. The Steiner Tree Problem I: Formulations, Compositions and Extension of Facets, *Math. Programming* **64** 209–229.

Dror, M., M. Haouari, J. Chaouachi. 2000. Generalized Spanning Trees. *Eur. J. of Oper. Res.* **120** 583–592.

Feremans, C. 2001. *Generalized Spanning Tree and Extensions*, Ph.D. thesis, Université Libre de Bruxelles.

Feremans, C., M. Labbé, G. Laporte. 2002. A Comparative Analysis of Several Formulations for the Generalized Minimum Spanning Tree Problem. *Networks* **39** 29–34.

Feremans, C., M. Labbé, G. Laporte. 2004. The Generalized Minimum Spanning Tree Problem: Polyhedral Analysis and Branch-and-Cut Algorithm. *Networks* **43** 71–86.

Fischetti, M., J.J. Salazar González, P. Toth. 1997. A Branch-and-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem. *Oper. Res.* **45** 378–394.

Glover, F. 1986. Future Paths for Integer Programming and Links to Artificial Intelligence. *Comput. & Oper. Res.* **13** 533–549.

Glover, F., M. Laguna. 1997. *Tabu Search*, Kluwer, Dordrecht.

Golden, B.L., S. Raghavan, D. Stanojević. 2005. Heuristic Search for the Generalized Minimum Spanning Tree Problem. *INFORMS J. on Computing* **17** 290–304.

Golden, B.L., S. Raghavan, D. Stanojević. 2006. Private communication.

Haouari, M., J.S. Chaouachi. 2002. A Probabilistic Greedy Search Algorithm for Combinatorial Optimization with Application to the Set Covering Problem, *J. of the Oper. Res. Soc.* **53** 792–799.

Haouari, M., J.S. Chaouachi. 2006. Upper and Lower Bounding Strategies for the Generalized Minimum Spanning Tree Problem, *Eur. J. of Oper. Res.* **172** 632–647.

Prim, R.C. 1957. Shortest connection networks and some generalizations. *Bell System Technical Journal* **36** 1389–1401.

Kruskal, J.B. 1956. On the shortest spanning tree of graph and the traveling salesman problem. *Proceedings of the American Mathematical Society* **7** 48–50.

Myung, Y.S., C.H. Lee, D.W. Tcha. 1995. On the Generalized Minimum Spanning Tree Problem, *Networks* **26** 231–241.

Pop, P.C., W. Kern, G. Still. 2006. A New Relaxation Method for the Generalized Minimum Spanning Tree Problem, *Eur. J. of Oper. Res.* **170** 900–908.

Reinelt, G. 1991. TSPLIB–A Traveling Salesman Problem Library. *INFORMS J. on Computing* **3** 376–384.

Taillard, É.D. 1993. Parallel Iterative Search Methods for Vehicle Routing Problems, *Networks* **23** 661–673.

Takahashi, H., A. Matsuyama. 1980. An Approximate Solution for the Steiner Problem in Graphs. *Mathematica Japonica* **24** 573–577.

Table 1: Computational tesults for LS, GA1 and our TS for TSPLIB instances with sizes $198 \leq n \leq 226$. Optimal solution values are in bold-face.

		Lower	LS	GA	TS	
Instances	k	**Bound**	**Cost**	**Cost**	**Cost**	**Secs.**
Center clustering						
d198	40	7044	**7044**	**7044**	**7044**	31
gr202	41	242	244	243	**242**	33
ts225	45	62246	62400	62315	62269	37
pr226	46	55515	**55515**	**55515**	**55515**	39
$\mu = 3$						
d198	67	8283	8285	**8283**	**8283**	58
gr202	68	292.5	293	293	293	64
ts225	75	79019	79085	**79019**	**79019**	70
pr226	84	62527	**62527**	**62527**	**62527**	84
$\mu = 5$						
d198	40	7098	**7098**	**7098**	**7098**	31
gr202	41	232	234	**232**	**232**	34
ts225	45	60578	60713	60659	**60588**	37
pr226	50	56721	**56721**	**56721**	**56721**	43
$\mu = 7$						
d198	32	6501	**6501**	**6501**	**6501**	23
gr202	31	202	203	203	203	23
ts225	35	50813	**50813**	**50813**	**50813**	26
pr226	33	48249	**48249**	**48249**	**48249**	26
$\mu = 10$						
d198	25	6185	**6185**	**6185**	**6185**	17
gr202	21	177	**177**	**177**	**177**	14
ts225	25	40339	**40339**	**40339**	**40339**	18

Table 2: Computational results with the LS, GA1, GA2 (Golden et al. 2006) and our TS on new TSPLIB instances with the center clustering procedure.

		LS		GA1		GA2		TS	
Instances	k	**Cost**	**Secs.**	**Cost**	**Secs.**	**Cost**	**Secs.**	**Cost**	**Secs.**
ali535	107	114350	374	114379	492	**114303**	595	**11403**	683
att532	107	12017	401	12007	500	**12001**	604	**12001**	597
d493	99	16512	295	16526	388	**16493**	506	**16493**	587
d657	132	19487	942	19465	969	19433	810	**19427**	1056
fl417	84	7936	133	7936	218	**7935**	142	**7935**	233
gil262	53	890	40	**887**	73	**887**	41	**887**	74
gr229	46	**59740**	27	**59740**	39	**59740**	25	**59740**	63
gr431	87	86923	218	86899	266	86891	200	**86885**	233
gr666	134	145015	797	144918	2866	**144756**	2296	**144756**	1365
lin318	64	18488	77	18476	105	**18471**	75	**18471**	130
p654	131	22218	641	22214	881	22209	678	**22208**	1045
pa561	113	870	1114	868	559	**864**	425	**864**	702
pcb442	89	19656	224	19670	284	**19571**	373	**19571**	266
pr264	53	**21872**	38	21886	57	**21872**	40	**21872**	72
pr299	60	20291	64	20307	86	**20290**	65	**20290**	94
pr439	88	51799	203	51808	981	51763	835	**51760**	574
rat575	115	2187	603	2189	627	**2170**	542	**2170**	762
rat783	157	3043	1307	3044	1653	3021	1432	**3017**	1916
rd400	80	5884	155	5880	205	5869	171	**5868**	208
si535	107	**12791**	302	**12791**	458	**12791**	487	**12791**	573
u574	115	15073	603	15069	620	**15037**	541	**15037**	517
u724	145	16039	1076	16015	1281	**15905**	1052	**15905**	1290
Average		**30594.6**	**437**	**30589.7**	**618**	**30557.8**	**542**	**30556.9**	**690**

Table 3: Computational results with the LS, GA1, GA2 (Golden et al. 2006) and our TS on new TSPLIB instances with the grid clustering procedure $\mu = 3$.

		LS		GA1		GA2		TS	
Instances	k	**Cost**	**Secs.**	**Cost**	**Secs.**	**Cost**	**Secs.**	**Cost**	**Secs.**
ali535	181	134468	1130	**134362**	1449	134370	1021	**134362**	1576
att532	182	15664	1102	15662	1437	**15652**	988	**15652**	1569
d493	171	20282	772	**20269**	1058	**20269**	699	**20269**	1104
d657	221	25781	2054	25718	2541	25714	1741	**25703**	3027
fl417	142	8358	341	8355	2368	8355	1503	**8353**	681
gil262	95	**1255**	120	**1255**	178	**1255**	202	**1255**	204
gr229	81	**74792**	87	**74792**	128	**74792**	72	**74792**	67
gr431	149	**103844**	563	**103844**	2253	**103844**	1406	**103844**	1697
gr666	224	174800	2055	174683	2623	174681	1802	**174671**	3105
lin318	108	**24092**	209	**24092**	257	**24092**	167	**24092**	206
p654	230	23561	1551	**23553**	2641	23555	5334	**23553**	4127
pcb442	156	27638	585	27525	813	**27513**	581	**27513**	810
pr264	101	**29199**	108	**29199**	1228	**29199**	128	**29199**	102
pr299	102	**23096**	155	**23096**	1108	**23096**	702	**23096**	399
pr439	163	65048	1695	64996	824	64983	543	**64953**	932
rat575	196	2942	2854	2898	1671	2896	1192	**2884**	2080
rat783	285	4253	4823	4251	6081	**4211**	4161	**4211**	7180
rd400	135	7651	430	7642	646	7638	1094	**7632**	548
u574	198	19743	2928	**19700**	1555	19701	1054	**19699**	1340
u724	266	21854	3408	21823	4340	**21761**	3002	**21761**	5201
Average		**40416.1**	**1348**	**40385.8**	**1759**	**40378.9**	**1369**	**40374.8**	**2042**

Table 4: Computational results with the LS, GA1, GA2 (Golden et al. 2006) and our TS on new TSPLIB instances with the grid clustering procedure $\mu = 5$.

		LS		**GA1**		**GA2**		**TS**	
Instances	k	**Cost**	**Secs.**	**Cost**	**Secs.**	**Cost**	**Secs.**	**Cost**	**Secs.**
ali535	108	**108201**	409	**108201**	497	**108201**	562	**108201**	632
att532	110	11903	420	11897	634	**11896**	661	**11896**	641
d493	102	16155	284	**16132**	383	16143	519	**16132**	725
d657	137	19914	952	19878	980	19854	763	**19846**	1249
fl417	93	**7952**	153	7953	250	**7952**	163	**7952**	230
gil262	63	989	55	988	472	985	62	**984**	63
gr229	47	**54305**	31	**54305**	41	**54305**	26	**54305**	43
gr431	87	**81856**	214	**81856**	250	81872	170	**81856**	244
gr666	139	144235	739	**144077**	3490	**144077**	2216	**144077**	2192
lin318	64	**17667**	66	**17667**	92	**17667**	59	**17667**	116
p654	134	22390	506	22380	781	22383	569	**22379**	1316
pcb442	95	19604	221	19461	297	19420	405	**19411**	357
pr264	55	**21351**	34	**21351**	57	**21351**	36	**21351**	67
pr299	69	18632	86	**18582**	107	**18582**	77	**18582**	105
pr439	96	54253	251	54253	1547	54253	976	**54230**	587
rat575	121	2065	682	2036	661	2024	533	**2018**	705
rat783	169	2912	1594	2877	2262	**2832**	1420	**2832**	2424
rd400	81	5461	170	5450	210	**5435**	189	**5434**	284
u574	127	15313	653	15289	627	15259	507	**15255**	1037
u724	166	15565	1334	15615	1459	**15458**	1075	**15458**	2064
Average		**32036.2**	**442**	**32012.4**	**754**	**31997.5**	**549**	**31993.3**	**814**

Table 5: Computational results with the LS, GA1, GA2 (Golden et al. 2006) and our TS on new TSPLIB instances with the grid clustering procedure $\mu = 7$.

		LS		GA1		GA2		TS	
Instances	k	**Cost**	**Secs.**	**Cost**	**Secs.**	**Cost**	**Secs.**	**Cost**	**Secs.**
ali535	83	**94149**	242	**94149**	1840	**94149**	962	**94149**	1012
att532	80	10231	214	10211	1499	10214	1063	**10206**	992
d493	78	14174	169	**14152**	225	**14152**	176	**14152**	212
d657	96	16663	479	16586	489	16578	422	**16542**	440
fl417	61	**7446**	69	**7446**	113	**7446**	74	**7446**	99
gil262	49	803	38	803	49	803	35	**802**	63
gr229	34	**46049**	16	**46049**	22	**46049**	14	**46049**	13
gr431	64	71416	114	71416	138	71416	97	**71415**	111
gr666	96	119783	354	**119271**	446	**119271**	309	**119271**	452
lin318	49	**14909**	43	**14909**	61	**14909**	39	**14909**	44
p654	100	21773	2012	21771	460	**21770**	321	**21770**	462
pcb442	64	14753	105	14709	996	**14644**	102	**14644**	117
pr264	43	**20438**	23	**20438**	37	**20438**	24	**20438**	28
pr299	47	15271	38	15241	55	**15238**	36	**15238**	37
pr439	74	**47101**	151	**47101**	217	**47101**	141	**47101**	140
rat575	100	1762	529	1745	475	1743	382	**1735**	601
rat783	121	2271	897	2238	1001	**2230**	747	**2230**	850
rd400	64	4595	109	4602	135	**4581**	102	**4581**	97
u574	92	12230	290	12191	349	**12186**	265	**12186**	304
u724	119	13305	912	13180	755	13151	645	**13101**	710
Average		**27456.1**	**340**	**27410.4**	**468**	**27403.5**	**297**	**27398.3**	**357**

Table 6: Computational results with the LS, GA1, GA2 (Golden et al. 2006) and our TS on new TSPLIB instances with the grid clustering procedure $\mu = 10$.

		LS		GA1		GA2		TS	
Instances	k	**Cost**	**Secs.**	**Cost**	**Secs.**	**Cost**	**Secs.**	**Cost**	**Secs.**
ali535	57	73376	111	**73359**	146	73376	100	**73359**	213
att532	57	**8497**	109	**8497**	151	**8497**	105	**8497**	187
d493	52	**11121**	83	**11121**	141	**11121**	84	**11121**	180
d657	73	13531	244	**13495**	318	13530	275	**13495**	386
fl417	42	**6986**	35	**6986**	54	**6986**	36	**6986**	75
gil262	36	**639**	19	**639**	27	**639**	18	**639**	33
gr229	23	**39793**	8	**39793**	16	**39793**	7	**39793**	28
gr431	52	**62722**	73	**62722**	95	**62722**	64	**62722**	110
gr666	70	97229	208	**97198**	251	**97198**	172	**97198**	315
lin318	36	**10119**	352	**10119**	37	**10119**	24	**10119**	31
p654	69	**20739**	150	**20739**	222	**20739**	149	**20739**	274
pcb442	48	**11941**	74	**11941**	102	**11941**	58	**11941**	135
pr264	27	**16546**	22	**16546**	17	**16546**	11	**16546**	28
pr299	35	**11624**	25	**11624**	32	**11624**	21	**11624**	63
pr439	48	**40518**	338	**40518**	80	**40518**	53	**40518**	117
rat575	64	**1235**	182	**1235**	206	**1235**	148	**1235**	284
rat783	81	1711	563	1697	445	**1682**	330	**1682**	468
rd400	49	3844	69	3835	78	**3825**	60	**3825**	99
u574	64	9758	146	9759	185	**9755**	135	**9755**	234
u724	80	9703	316	9633	365	**9608**	294	**9608**	365
Average		**22581.6**	**156**	**22572.8**	**148**	**22572.7**	**107**	**22570.1**	**181**

Table 7: Computational results with the GA (Haouari and Chaouachi 2006) and our TS for the L-GMSTP

Instances	Optimal solution cost	GA		TS	
		Cost	Secs.	Cost	Secs.
DHC1	**23**	**23**	0.02	**23**	0.08
DHC2	**41**	**41**	0.01	**41**	0.06
DHC3	**36**	**36**	0.03	**36**	0.1
DHC4	**18**	**18**	0.01	**18**	0.6
DHC5	**27**	**27**	0.11	**27**	0.41
DHC6	**55**	**55**	0.12	**55**	0.48
DHC7	**67**	**67**	0.2	**67**	0.59
DHC8	**53**	**53**	0.3	**53**	0.68
DHC9	**37**	**37**	0.26	**37**	0.63
DHC10	**48**	**48**	1.64	**48**	2.43
DHC11	**50**	**50**	0.46	**50**	1.01
DHC12	**68**	**68**	3.24	**68**	7.03
DHC13	**44**	**44**	2.36	**44**	5.38
DHC14	**50**	**50**	7.13	**50**	16.9
DHC15	**60**	**60**	8.25	**60**	19.6
DHC16	**117**	118	45	118	134
DHC17	**88**	89	115	89	269
DHC18	**85**	86	246	86	395
DHC19	**88**	91	331	94	594
DHC20	**105**	120	379	124	617
Average	**58**	**59.05**	**57**	**59.40**	**120**